

BUNDESREPUBLIK DEUTSCHLAND

PCT/EP2004/004991

10.05.2004



REC'D 28 MAY 2004
WIPO PCT

PRIORITY
DOCUMENT
SUBMITTED OR TRANSMITTED IN
COMPLIANCE WITH RULE 17.1(a) OR (b)

Prioritätsbescheinigung über die Einreichung
einer Patentanmeldung

Aktenzeichen:

103 29 147.4

Anmeldestag:

27. Juni 2003

Anmelder/Inhaber:

Siemens Aktiengesellschaft, 80333 München/DE

Bezeichnung:

Verknüpfung und Darstellung von Signalen einer
Vorrichtung zur Hardware-Simulation und Elementen
eines Listings eines Programms

IPC:

G 06 F 19/00

Die angehefteten Stücke sind eine richtige und genaue Wiedergabe der ur-
sprünglichen Unterlagen dieser Patentanmeldung.

München, den 04. März 2004
Deutsches Patent- und Markenamt
Der Präsident
Im Auftrag

Wallner

Beschreibung

5 Verknüpfung und Darstellung von Signalen einer Vorrichtung zur Hardware-Simulation und Elementen eines Listings eines Programms

10 Die Erfindung betrifft ein System sowie ein Verfahren zur Darstellung von Signalen einer Vorrichtung zur Hardware-Simulation sowie ein Fehlersuchwerkzeug.

15 Aus US 5 768 567 ist ein Hardware-Software-Cosimulator zur Simulation eines Hardware-Software-Systems bekannt, welcher einen logischen Simulator, Busschnittstellenmodelle, Speichermodelle, Befehlssatzsimulatoren und einen sogenannten Cosimulations-Optimierungs-Manager aufweist. Die gleichzeitige Simulation von Hardware und Software wird auch Cosimulation genannt. Dabei wird die Cosimulation mit einem einzigen kohärenten Blick auf den Speicher des Hardware-Software-Systems durchgeführt, welcher durch den Cosimulations-Optimierungs-Manager sowohl für Hardware- als auch für Software-Simulationen transparent aufrechterhalten wird.

20 5 Der Erfindung liegt die Aufgabe zugrunde, die gemeinsame Darstellung von Signalen einer Vorrichtung zur Hardware-Simulation und Elementen eines Listings eines Programms zu ermöglichen.

25 Diese Aufgabe wird durch ein System zur Verknüpfung und Darstellung von Signalen einer Vorrichtung zur Hardware-Simulation und Elementen eines Listings eines Programms gelöst,

30 35

- wobei die Vorrichtung zur Hardware-Simulation das Verhalten einer Schaltung mit einem Prozessor, einem Programm-Speicher, welcher den Programmcode des Programms enthält, und applikationsspezifischen Hardwarekomponenten simuliert und Signale als Ergebnis der Simulation erzeugt,

- wobei die Elemente des Listings des Programms mit den bei der simulierten Ausführung des im Programmspeicher enthaltenen, mit diesen Elementen korrespondierenden Programmcode des erzeugten Signalen verknüpft werden,

5

- wobei die Elemente des Listings des Programms in einem ersten Teilbereich eines grafischen Anzeigemittels und die Signale in einem zweiten Teilbereich des Anzeigemittels darstellbar sind.

10

Diese Aufgabe wird durch ein Verfahren zur Verknüpfung und Darstellung von Signalen einer Vorrichtung zur Hardware-Simulation und Elementen eines Listings eines Programms gelöst,

- wobei die Vorrichtung zur Hardware-Simulation das Verhalten einer Schaltung mit einem Prozessor, einem Programmspeicher, welcher den Programmcode des Programms enthält, und applikationsspezifischen Hardwarekomponenten simuliert und Signale als Ergebnis der Simulation erzeugt,

15

- wobei die Elemente des Listings des Programms mit den bei der simulierten Ausführung des im Programmspeicher enthaltenen, mit diesen Elementen korrespondierenden Programmcode des erzeugten Signalen verknüpft werden,

20

- wobei die Elemente des Listings des Programms in einem ersten Teilbereich eines grafischen Anzeigemittels und die Signale in einem zweiten Teilbereich des Anzeigemittels dargestellt werden.

30

Diese Aufgabe wird durch ein Fehlersuchwerkzeug zur Verknüpfung und Darstellung von Signalen einer Vorrichtung zur Hardware-Simulation und Elementen eines Listings eines Programms gelöst,

- wobei die Vorrichtung zur Hardware-Simulation das Verhalten einer Schaltung mit einem Prozessor, einem Programmspeicher, welcher den Programmcode des Programms enthält, und applikationsspezifischen Hardwarekomponenten simuliert und Signale als Ergebnis der Simulation erzeugt,

- o wobei das Fehlersuchwerkzeug Mittel zur Verknüpfung der Elemente des Listings des Programms mit den bei der simulierten Ausführung des im Programmspeicher enthaltenen, mit diesen Elementen korrespondierenden Programmcodes erzeugten Signalen aufweist,

5 wobei die Elemente des Listings des Programms in einem ersten Teilbereich eines grafischen Anzeigemittels und die Signale in einem zweiten Teilbereich des Anzeigemittels darstellbar sind.

10

Der Erfindung liegt die Erkenntnis zugrunde, dass das Design von Hardware-Software-Systemen durch eine verknüpfte Darstellung der Signale eines Hardware-Simulators und der Elemente des Listings eines Programms wesentlich vereinfacht wird. Das erfindungsgemäße System und Verfahren ermöglicht eine einfache und fehlersichere Verfolgung der Ausführung eines Programms. Die Darstellung des Listings des Programms stellt einen direkten Bezug zum eigentlichen Programm her, insbesondere können, auch aufgrund der Verwendung des Listfiles die im Original-Quelltext eingefügten Kommentare angezeigt werden. Zur Verknüpfung und Darstellung der Signale der Vorrichtung zur Hardware-Simulation und der Elemente des Listings des Programms ist das Vorliegen eines abstrahierten Software-Modells des Prozessors nicht erforderlich. Es ist somit gewährleistet, dass tatsächlich der in der Schaltung verwendete Prozessor simuliert wird. Somit werden durch eine Beschreibung des Prozessors mittels eines Modells verursachte Fehler bei der Simulation ausgeschlossen. Die Visualisierung des Programmablaufs erfolgt in ähnlicher Form wie bei üblichen Entwicklungswerkzeugen zum Debuggen von reiner Software. Die Einarbeitungszeit für Anwender des Systems bzw. des Verfahrens, welche Erfahrung bei der Entwicklung von Software oder Firmware haben, ist daher relativ gering.

30

35 Gemäß einer vorteilhaften Ausgestaltung der Erfindung ist eine Markierung eines Elements des Listings des Programms im ersten Teilbereich des grafischen Anzeigemittels und eine

Markierung der mit diesem Element verknüpften Signale im zweiten Teilbereich des Anzeigemittels vorgesehen. Die grafische Darstellung orientiert sich somit an üblichen Software-Debug-Werkzeugen. Der Programmverlauf kann durch eine Markierung eines Elements des Listings des Programms verfolgt werden. Im zweiten Teilbereich des Anzeigemittels bewegt sich eine Markierung synchronisiert zur Markierung im ersten Teilbereich des Anzeigemittels, so dass auch Querbeziehungen zur übrigen Hardware erfasst werden können.

10

Vorteilhafterweise ist ein dritter Teilbereich des grafischen Anzeigemittels zur Darstellung mindestens eines Teils der Signale, insbesondere weiterer Werte, vorgesehen. Weitere Werte können beispielsweise Registerwerte sein.

15

Die Verwendung üblicher Vorrichtungen zur Hardware-Simulation wird erleichtert, wenn gemäß einer vorteilhaften Ausgestaltung der Erfindung die Schaltung mit dem Prozessor, dem Programmspeicher und den applikationsspezifischen Hardwarekomponenten in einer Hardware-Beschreibungssprache beschrieben sind.

20

Das System lässt sich mit geringem Aufwand an verschiedene Prozessoren anpassen, wenn gemäß einer vorteilhaften Ausgestaltung der Erfindung Mittel zum Anpassen des Systems an verschiedene Prozessortypen vorgesehen sind.

Nachfolgend wird die Erfindung anhand der in den Figuren dargestellten Ausführungsbeispiele näher beschrieben und erläutert.

30

Es zeigen:

35

FIG 1 eine schematische Darstellung eines Systems zur Verknüpfung und Darstellung von Signalen einer Vorrichtung zur Hardware-Simulation und Elementen eines Listings eines Programms,

FIG 2 einen Ausschnitt einer Darstellung von Signalen einer Vorrichtung zur Hardware-Simulation und

5 FIG 3 eine Darstellung von Signalen einer Vorrichtung zur Hardware-Simulation und Elementen eines Listings eines Programms.

10 FIG 1 zeigt ein System zur Verknüpfung und Darstellung von Signalen 4, 5 einer Vorrichtung zur Hardware-Simulation 1 und Elementen 15 eines Listings 2 eines Programms 3 in schematischer Darstellung. Die Vorrichtung zur Hardware-Simulation 1 simuliert das Verhalten einer Schaltung 6 mit einem Prozessor

15 7, einem Programmspeicher 8 und applikationsspezifischen Hardwarekomponenten 10. Der Prozessor 7 kann z. B. ein Mikroprozessor oder Mikrocontroller sein. Der Programmspeicher 8 enthält den Programmcode 9 des Programms 3. Die Schaltung 6 wird mit Hilfe einer Hardware-Beschreibungssprache, abgekürzt

20 HDL (HDL = Hardware Description Language), im Ausführungsbeispiel mittels VHDL (VHDL = Very High Speed Integrated Circuit Hardware Description Language) beschrieben. Mit dem Bezugzeichen 17 wird der Quellcode in VHDL bezeichnet. Die Vorrichtung zur Hardware-Simulation 1 wird entsprechend auch als HDL-Simulator bezeichnet. Ein Beispiel für einen HDL-

25 Simulator ist das Produkt "ModelSim" der Firma Model Technology, Portland, Oregon, USA. Die Schaltung 6 ist z. B. ein sogenannter ASIC (ASIC = Application Specific Integrated Circuit = Anwendungsspezifische integrierte Schaltung). Der HDL-Quellcode 17 wird mit einem geeigneten Compiler in ein HDL-

30 Simulator-Format 18 umgewandelt, welches von der Vorrichtung zur Hardware-Simulation 1 verarbeitet werden kann. Die Vorrichtung zur Hardware-Simulation 1 erzeugt Signale 4, 5 als Ergebnis der Simulation. Das Programm 3, bzw. der Programmcode 9 des Programms 3 wird mittels eines Compilers in ein Da-

35 tenfile 16 (üblicherweise mit Daten in hexadezimaler Form) und ein Listing 2, auch Listfile genannt, umgewandelt. Das Listing 2 enthält sowohl die Programmbefehle als auch die zu-

gehörigen Kommentare. Das Listing 2 ist zeilenweise aufgebaut, wobei jede Zeile jeweils einen Programmbebefl bzw. eine Anweisung enthält, gegebenenfalls mit dem zugehörigen Kommentar. Eine Zeile, ein Programmbebefl, eine Anweisung bzw. ein Kommentar sind Elemente 15 des Listings 2. Ein Debugger 19 verknüpft die Elemente 15 des Listings 2 des Programms 3 mit den bei der simulierten Ausführung des im Programmspeicher 8 enthaltenen, mit diesen Elementen 15 korrespondierenden Programmcodes 9 erzeugten Signalen 4, 5. Die Elemente 15 des Listings 2 des Programms 3 werden in einem ersten Teilbereich 11 des grafischen Anzeigemittels 14 und die Signale 4, 5 in einem zweiten Teilbereich 12 bzw. in einem dritten Teilbereich 13 des Anzeigemittels 14 dargestellt.

15 FIG 2 zeigt einen Ausschnitt 30 einer Darstellung von Signalen 4, 5 einer Vorrichtung zur Hardware-Simulation 1. Die Signale 4, 5 werden als sogenannte Waveforms 35, 36 und 37 dargestellt.

20 FIG 3 zeigt eine Darstellung 23 von Signalen 4, 5 einer Vorrichtung zur Hardware-Simulation 1 und Elementen 15 eines Listings 2 eines Programms 3. Die Elemente 15 des Listings 2 werden in einem ersten Teilbereich 11, die Signale 4, 5 in einem zweiten 12 bzw. einem dritten 13 Teilbereich eines Anzeigemittels 14 dargestellt. Die Elemente 15 können mit einer Markierung 20, z. B. einer farblichen Kennzeichnung, versehen werden. Die Signale 4 können mit einer Markierung 21, z. B. einem Cursor, gekennzeichnet werden. Ein Anzeigemittel 14 kann eine Bildschirmoberfläche sein, die Teilbereiche 11 bis 13 können als Anzeigefenster realisiert sein. Der Ablauf des Programms 3 im Prozessor 7 (siehe FIG 1) wird mit Hilfe des grafischen Frontends, des Debuggers 19, welcher auf die Signale 4, 5 der Vorrichtung zur Hardware-Simulation 1 aufsetzt, visualisiert. Die Visualisierung des Programmablaufes erfolgt durch Zugriff auf Signale des Prozessors 7, deren Werte durch die Vorrichtung zur Hardware-Simulation 1 berechnet wurden.

Gemäß dem Ausführungsbeispiel ist der Debugger 19 in der Skriptsprache TCL/TK realisiert (TCL/TK = Tool Command Language/Toolkit) und benutzt die TCL/TK-Schnittstelle zu HDL-Objekten, die ein HDL-Simulator zur Verfügung stellt. Solche

5 HDL-Objekte können z. B. als Waveforms dargestellt werden. Um den Debugger 19 möglichst flexibel zu gestalten, d. h. in diesem Zusammenhang, dass er relativ einfach an verschiedene Prozessoren angepasst werden kann, wird der Debugger 19 in zwei Abschnitte aufgeteilt. Ein erster allgemeiner Teil

10 stellt Prozeduren zur Visualisierung bereit. Ein zweiter prozessorspezifischer Teil ist dem jeweiligen Prozessor anpassbar und setzt sich z. B. aus folgenden Prozeduren zusammen:

- 15 ◦ Prozeduren für Single-Step rechts/links
- Prozeduren zum Bestimmen der Registerwerte
- Prozeduren für die Kopplung von Signalen und Elementen des Listings
- Prozeduren zur Konfiguration

20 In den Prozeduren für Single-Step rechts/links (Bezugszeichen 22 in FIG 3) wird der Cursor in einem Waveform-Fenster auf den nächsten gültigen Befehl gesetzt. Für das Beispiel eines KRISC8-Prozessors (KRISC8 = Kommunikations-RISC 8 Bit, Spezial-Core für die Kommunikation in Feldbusssystemen der Firma Siemens AG, München, Deutschland) ist dies in FIG 2 veranschaulicht.

5 Der im Folgenden wiedergegebene Ausschnitt aus einem Listing zeigt eine Prozedur für Single-Step-Right.

```

# Prozedur für Single-Step-Right
# ACHTUNG: pc muss im Wave-Fenster selektiert sein (Das right-Kommando wirkt auf das
#          selektierte Signal)!!!!!
proc right_krisc8.Proc {} {
5    global minideb
    global DEBUG

    set time_now [lindex [getactivecursortime] 0]
    set address [examine -time $time_now $minideb(TIME_UNIT) -hex $minideb(pc-path)]
    set address_tmp [examine -time [expr $time_now + $minideb(CLK_PERIOD)] $minideb(TIME_UNIT) -hex
10   $minideb(pc-path)]
    set i 1

    if {$DEBUG == "on"} {echo "anfang von right.Proc"}
    if {$address_tmp == "No_Data"} {
15      .mainFrame.label_message configure -bg DimGray -fg red -text "End of Simulation reached."
      set minideb(auto_step_right) 0
      .mainFrame.frame_toolbar.autostep_right configure -background grey -activebackground khaki2
      .mainFrame.frame_toolbar.label_info configure -text ""
      return
    }

20    # Anfang vom naechsten Befehl suchen
    while {$address == $address_tmp} {
      if {$DEBUG == "on"} {echo "right.Proc: while-Schleife 1"}
      incr i

      set address_tmp [examine -time [expr $time_now + $i * $minideb(CLK_PERIOD)] $minideb(TIME_UNIT)
-hex $minideb(pc-path)]
      }

      set time_now [expr $time_now + $i * $minideb(CLK_PERIOD)]
      if {$DEBUG == "on"} {echo "right.Proc:time_now --> $time_now"}
      set schleife 1
    }

30    # Suche den naechseten gueltigen Befehl
    while {$schleife == 1} {
      if {$DEBUG == "on"} {echo "right.Proc: while-Schleife 2"}
      # Ende des Befehls suchen
      set address [examine -time $time_now $minideb(TIME_UNIT) -hex $minideb(pc-path)]
      set address_tmp [examine -time [expr $time_now + $minideb(CLK_PERIOD)] $minideb(TIME_UNIT) -hex
35   $minideb(pc-path)]
      if {$address_tmp == "No_Data"} {

```

```

.mainFrame.label_message configure -bg DimGray -fg red -text "End of Simulation reached."
set minideb(auto_step_right) 0
.mainFrame.frame_toolbar.autostep_right configure -background grey -activebackground khaki2
.mainFrame.frame_toolbar.label_info configure -text ""
5      return
}

set i 1
while {$address == $address_tmp} {
    if {$DEBUG == "on"} {echo "right.Proc: while-Schleife 3"}
10     incr i

    set address_tmp [examine -time [expr $time_now + $i * $minideb(CLK_PERIOD)] $minideb(TIME_UNIT) -hex $minideb(pc-path)]
}

set time_now_tmp [expr $time_now + $i * $minideb(CLK_PERIOD)]
15 # Befehl ist 16 Bit breit?

set instruction [examine -time $time_now $minideb(TIME_UNIT) -hex $minideb(instruction)]
set digit [split $instruction ""]
set k1 "[lindex $digit 0][lindex $digit 1]"
if {$DEBUG == "on"} {echo "right.Proc: splitted instruction --> <$k1>"}
20 if {$k1 == "1E"} {

    # Befehl ist 16 Bit breit, 1 Befehl weitergehen
    if {$DEBUG == "on"} {echo "right.Proc: 16 Bit Befehl"}
    set time_now $time_now_tmp
} else {

    set read_time [expr $time_now_tmp - 0.5 * $minideb(CLK_PERIOD)]
    if {$DEBUG == "on"} {echo "right.Proc: time_now --> $time_now; time_now_tmp --> $time_now_tmp; read_time --> $read_time"}

    if {$DEBUG == "on"} {echo "en_pipe --> [examine -time $read_time $minideb(TIME_UNIT) -bin $minideb(en_pipe)]; \
30          res_pipe --> [expr !{examine -time $read_time $minideb(TIME_UNIT) -bin $minideb(res_pipe)}]"}
        if {[examine -time $read_time $minideb(TIME_UNIT) -bin $minideb(en_pipe)] && !{examine -time $read_time $minideb(TIME_UNIT) -bin $minideb(res_pipe)}} {
            # naechsten gueltigen Befehl gefunden
35            set schleife 0
        } else {
            if {$DEBUG == "on"} {echo "right.Proc: Der Befehl ist ungültig!"}
        }
    }
}

```

```
10
# Befehl wird nicht ausgefuehrt, da
# - pipe enable nicht gesetzt ist
# - pipe reset gesetzt ist
# 1 Befehl weiter gehen
5      set time_now $time_now_tmp
}
}
}
set address_int [examine -time $time_now $minideb(TIME_UNIT) -hex $minideb(pc-path)]
10 if {[catch {.$minideb(wave_name).tree right -value 'h$address_int 1} result] !=0} {
# Wenn PC nicht markiert ist, deaktiviere Auto-Step und zeige eine Fehlermeldung an
set minideb(auto_step_right) 0
.mainFrame.frame_toolbar.autostep_right configure -background grey -activebackground khaki2
.mainFrame.frame_toolbar.label_info configure -text ""
15 notice_show "$result \nPlease select pc in the wave-Window!"
}
trace.Proc
}
```

Dabei wird vereinfacht wie folgt vorgegangen (siehe FIG 2):

- Zum momentanen Simulationszeitpunkt wird der Wert des Programmcounters 36 (pc, Programmcounter = Programmzähler) ermittelt. Der momentan ausgeführte Befehl ist in FIG 2 mit dem Bezugszeichen 31 gekennzeichnet)
- Der Beginn des nächsten Befehles wird gesucht. Der Cursor wird solange um Vielfache der Taktperiode weitergeschoben, bis eine Änderung des Programmzählers auftritt.
- Es wird überprüft, ob der nun erreichte Befehl ausgeführt wird.
 - Handelt es sich um einen Befehl, welcher nicht ausgeführt wird, so wird der übernächste Befehl getestet (Wiederholung solange, bis ein ausgeführter Befehl erreicht ist oder das Simulationsende erreicht ist). Im Beispielsfall gemäß FIG 2 sind die nicht ausgeführten, d. h. ungültigen Befehle mit dem Bezugszeichen 32 gekennzeichnet.
 - Wird der erreichte Befehl ausgeführt, dann wird der Cursor im Waveform-Fenster (entspricht dem zweiten Teilbereich 12 des Anzeigemittels 14 gemäß FIG 3) zu diesem Zeitpunkt platziert. Im Beispielsfall gemäß FIG 2 ist der nächste ausgeführte, d. h. gültige Befehl mit dem Bezugszeichen 33 gekennzeichnet.

◦ Die Ermittlung des gültigen Befehls erfolgt prozessorspezifisch. Es genügt bei modernen Prozessorarchitekturen üblicherweise nicht, den Programcounter 36 zu verfolgen, sondern es sind zusätzliche Signale 4, 5 auszuwerten, die angeben, ob der momentane Befehl gültig ist (z. B. das der Waveform 34 zugrundeliegende Signal). Wenn ein gültiger Befehl ermittelt wurde, so erfolgt die Markierung 20 des momentan ausgeführten Befehls im angezeigten Listing im ersten Teilbereich 11 des Anzeigemittels 14 und es werden die Registerwerte im dritten Teilbereich 13 des Anzeigemittels 14 angezeigt (siehe folgenden Ausschnitt eines Listings).

```
proc trace.Proc {} {  
    global progadr_alt  
    global minideb  
    global DEBUG  
5     if {$minideb(configure_done) == 1} {  
        if {$minideb(cursor_move_enable) == 1} {  
            DisableCursorMove.Proc  
        }  
        # Programmzaehler auslesen  
10    set timenow [lindex [getactivecursortime] 0]  
        # aktuelle Programmadresse lesen  
        if {$DEBUG == "on"} {echo "trace.Proc: timenow eingelesen"}  
        set progadr [string tolower [examine -time $timenow $minideb(TIME_UNIT) -hex $minideb(pc-path)]]  
15    if {$DEBUG == "on"} {echo "trace.Proc: progadr gelesen <$progadr>"}  
        # Programmadresse ist undefiniert (Anfang von der Simulation) ?  
        if {[![regexp {[0-9a-fA-F]+} $progadr]} {  
            set progadr 0  
        }  
20    # gelesenen Programmcounter mit 2 multiplizieren, um die Programmadresse zu erhalten (bei NIOS)  
        # vorangestellte Nullen werden abgeschnitten  
        # da mit hex-Zahlen nicht gerechnet werden kann, wird auf eine Dezimalzahl umgerechnet,  
        # das Ergebnis mit 2 multipliziert (bei NIOS, 1 bei KRISC) und anschließend wieder in eine  
        # hex-Zahl konvertiert.  
        set progadr_int [hex2int $progadr]  
        set progadr_int [expr $minideb(adressmultiplikator)*$progadr_int]  
        set progadr [int2hex $progadr_int]  
  
        SearchCodeLine.Proc $progadr  
30    GetRegisterValues_$minideb(mode).Proc  
        set progadr_alt $progadr  
    } else {  
        bell  
        .mainFrame.label_message configure -bg DimGray -fg red -text "Before tracing you have to config-  
35    re the program."  
    }  
}
```

Um den simulierten Ablauf eines Programms 3 auf einem Prozessor 7 zu verfolgen, gab es bisher keine zufriedenstellenden Möglichkeiten. Die manuelle Verfolgung des Programmablaufes anhand der von einem Simulator ausgegebenen Waveform und einem Ausdruck eines Listfiles hat den Nachteil, dass bei den üblicherweise verwendeten Prozessoren (Pipelines, Caches, u. Ä.) eine Programmverfolgung sehr aufwendig und fehlerträchtig wird. Bei einer Verfolgung des Programmablaufes anhand eines Disassemblerausdruckes muss ein Benutzer keine direkte Auswertung der Waveform vornehmen. Es wird nur eine Liste der vom Prozessor bearbeiteten Befehle ausgegeben. So mit fehlt aber der direkte Bezug zum eigentlichen Programm und damit werden auch die im Quelltext eingefügten Kommentare nicht angezeigt. Beim Verwenden eines Debuggers, der auf ein abstrahiertes C-Modell aufsetzt, ergeben sich im Wesentlichen zwei Nachteile. Zum einen benötigt man ein C-Modell für den eingesetzten Prozessor, welches normalerweise nicht vorliegt und somit aufwendig erstellt werden muss. Zum anderen stellt das C-Modell des Prozessors eine erneute Beschreibung des Prozessors dar und es ist nicht ausgeschlossen, dass der tatsächliche Prozessor ein vom C-Modell abweichendes Verhalten aufweist. Da bei dem hier vorgeschlagenen Verfahren kein C-Modell für den Prozessor 7 benötigt wird, ist zum einen gewährleistet, dass der gleiche Prozessor 7 simuliert wird, der auch in der Schaltung 6 implementiert ist. Zum anderen hält sich der Aufwand zum Anpassen des Debuggers 19 an verschiedene Prozessoren 7 bzw. Prozessortypen in Grenzen.

Zusammengefasst betrifft die Erfindung somit ein System und ein Verfahren zur Verknüpfung und Darstellung von Signalen 4, 30 5 einer Vorrichtung zur Hardware-Simulation 1 und Elementen 15 eines Listings 2 eines Programms 3 sowie ein Fehlersuch- werkzeug. Um eine gemeinsame Darstellung der Signale der Vor- rrichtung zur Hardware-Simulation und der Elemente des Lis- 35 tings des Programms zu ermöglichen, wird vorgeschlagen, dass die Vorrichtung zur Hardware-Simulation 1 das Verhalten einer Schaltung 6 mit einem Prozessor 7, einem Programmspeicher 8,

welcher den Programmcode 9 des Programms 3 enthält, und applikationsspezifischen Hardwarekomponenten 10 simuliert und Signale 4, 5 als Ergebnis der Simulation erzeugt, dass die Elemente 15 des Listings 2 des Programms 3 mit den bei der simulierten Ausführung des im Programmspeicher 8 enthaltenen, mit diesen Elementen 15 korrespondierenden Programmcodes 9 erzeugten Signalen 4, 5 verknüpft werden und dass die Elemente 15 des Listings 2 des Programms 3 in einem ersten Teilbereich 11 eines grafischen Anzeigemittels 14 und die Signale 4, 5 in einem zweiten Teilbereich 12 des Anzeigemittels 14 darstellbar sind.

Im Folgenden werden der technische Hintergrund der Erfindung und verwendete Begriffe näher erläutert.

Die beschriebenen Schaltungen werden z. B. in sogenannten eingebetteten Systemen (gebräuchlicher ist der entsprechende englische Begriff "embedded systems") eingesetzt. Beispiele für Programmiersprachen für den Einsatz in eingebetteten Systemen sind C, Assembler, C++ und Java. Diesen Sprachen gemeinsam ist der Einsatz eines Compilers, der eine schrittweise Vorgehensweise bei der Codeentwicklung vorschreibt, die sogenannten edit-compile-load-debug Zyklen. Debugger genannte Fehlersuchwerkzeuge werden allgemein verwendet, um Fehler in der Software-Programmierung aufzuspüren. Da die meisten Software-Entwicklungssysteme hierfür eine integrierte Entwicklungsumgebung zur Verfügung stellen, können Debugger relativ einfach den Programmcode verändern und mit Einzelschritten oder gesetzten Kontrollpunkten auf ihrem Zielsystem überprüfen. Man kann damit ein Programm kontrolliert ausführen, also das Programm Zeile für Zeile ausführen und die Werte von Variablen abfragen und ändern. Software Debugger bieten folgende Basisfunktionen an:

- Einzelschritt-Ausführung von Assembler- und Hochsprachencode

- Setzen von Haltepunkten (Breakpoints) an definierten Stellen in Hochsprache oder Maschinencode, an denen das Programm vorübergehend angehalten wird
- Anzeigen von Variablenwerten, logischen Ausdrücken, Speicheradressen und CPU-Registern

5

TCL ist eine plattformübergreifende Script-Programmiersprache. Durch die Kombination aus Textverarbeitungs-, Dateibearbeitungs- und Systemsteuerungsfunktionen ist TCL für diesen Zweck optimiert. EDA-Tools (EDA = Electronic Design Automation) verwenden TCL häufig zusammen mit dem Grafik-Toolkit TK, um eine flexible und plattformunabhängige grafische Benutzeroberfläche zu bieten. Hierzu gehört beispielsweise ModelSim.

15

Im Bereich der eingebetteten Systeme hat der parallele Entwurf von Hardware und Software den klassischen, rein sequentiellen Entwurfsablauf weitgehend abgelöst. Eingebettete Systeme spielen eine dominierende Rolle in der Automatisierungs-technik. Basis ihres Erfolgs ist die exponentiell steigende Komplexität integrierter Schaltungen und die damit wachsenden Möglichkeiten für neue Systemfunktionen, die zunehmend in Software realisiert werden. Die daraus resultierende System-komplexität verlangt bei gleichzeitig absinkenden Entwicklungszeiten eine drastische Erhöhung der Entwurfsproduktivität. Diese Erhöhung ist nur durch Entwurfsautomation und systematische Wiederverwendung von Systemfunktionen erreichbar. Ein wesentlicher Schritt ist die Integration von Hardware- und Softwareentwurf, der Hardware/Software-Coentwurf.

20

Hardware- und Softwareentwurf beginnen oft vor der Vollendung der Systemarchitektur oder gar vor der endgültigen Festlegung der Spezifikation. Systemarchitekten, Anwender und Kunden oder Marketingexperten entwickeln gemeinsam Anforderungsdefinition und Spezifikation. Der Systemarchitekt entwickelt daraus ein System kooperierender Systemfunktionen als Grundlage eines anschließenden, parallelen Entwurfs von Hardware und

35

Software. An dieser Stelle wird auch die Grobarchitektur der Zielhardware festgelegt. Der Hardware-/Software-Schnittstellenentwurf erfordert eine Beteiligung von Hardware- und Softwareentwicklern. Die Integration der Hardware und Softwarekomponenten und der Test des integrierten Systems folgt als letzter Schritt. In allen Phasen führen Abweichungen von erwarteten Entwurfsergebnissen oder Änderungen der Spezifikation zu einer Wiederholung von Entwurfsschritten. Ein zentrales Problem im Entwurfsprozess ist die Überwachung und Integration des parallelen Hardware- und Software-Entwurfs. Eine frühe Fehlererkennung erfordert die Kontrolle von Konsistenz und Korrektheit, die um so aufwendiger wird, je detaillierter der Entwurf ausgearbeitet ist.

Bei der Hardware-Software Cosimulation wird die Ausführung der Software auf der (Prozessor-)Hardware zusammen mit applicationsspezifischen Hardwarekomponenten simuliert. Da eine Simulation mit hohem Detaillierungsgrad, wie er im Hardwareentwurf üblich ist, zu langsam für eine praktisch nutzbare Softwaresimulation ist, werden üblicherweise abstrakte Prozessormodelle benötigt. Zu diesem Zweck werden die Prozessoren abstrakter ("auf einer höheren Abstraktionsebene") modelliert als die anderen Hardwarekomponenten. Dabei wird das Zeitverhalten des Prozessors nicht mehr taktgenau, d. h. für jeden Taktzyklus aufgeschlüsselt, dargestellt, sondern nur noch die Programme mit ihren Ein- und Ausgaben. Das Problem der Cosimulation liegt nun in der Kopplung der unterschiedlich abstrakten Modelle, derart dass eine hinreichende Genauigkeit der Simulation erreicht wird. Im ungünstigsten Fall greifen Prozessor und andere Hardwarekomponenten auf den gleichen Speicher zu. Eine genaue Modellierung erfordert in diesem Fall angepasste Speicher- und Busmodelle und spezielle Simulationstechniken. Ein Beispiel hierfür bietet der Cosimulator Seamless CVS der Firma Mentor Graphics, Wilsonville, Oregon, USA. Das Produkt Seamless CVS benutzt ein abstraktes Prozessormodell, das die Befehlausführung nachbildet (ein sogenannter Instruction Set Simulator). Für den Speicher-

zugriff werden Busmodelle verwendet, deren Abstraktion abhängig vom gleichzeitigen Zugriff von Prozessor und Hardware ist. Speicher, auf die lediglich der Prozessor zugreift, werden folglich abstrakter modelliert als solche, in denen Konflikte auftreten können. Voraussetzung ist also die Verfügbarkeit einer Bibliothek von Modellen, die vom CAD-Anbieter oder dem Prozessorhersteller bezogen wird.

Ein abstrakterer Ansatz reduziert das Prozessormodell auf die reine Programmausführung auf dem PC oder der Workstation und modelliert lediglich das Interface mit Zeitverhalten. Die Kopplung der Software-Ausführung zum Hardwaremodell erfolgt dann über ein simulatorspezifisches Kommunikationsprotokoll, das der Entwickler in die Software einfügen muss. Für diese Modellierung sind nur Interface-Modelle erforderlich, was die Bibliotheksproblematik wesentlich vereinfacht. Andererseits wird das Zeitverhalten nur auf der Hardwareseite korrekt modelliert. Beispiel für einen solchen Cosimulator ist das Produkt Eagle der Firma Synopsys, Mountain View, Kalifornien, USA.

Patentansprüche

1. System zur Verknüpfung und Darstellung von Signalen (4, 5) einer Vorrichtung zur Hardware-Simulation (1) und Elementen

5 (15) eines Listings (2) eines Programms (3),

◦ wobei die Vorrichtung zur Hardware-Simulation (1) das Verhalten einer Schaltung (6) mit einem Prozessor (7), einem Programmspeicher (8), welcher den Programmcode (9) des Programms (3) enthält, und applikationsspezifischen Hardwarekomponenten (10) simuliert und Signale (4, 5) als Ergebnis der Simulation erzeugt,

10

◦ wobei die Elemente (15) des Listings (2) des Programms (3) mit den bei der simulierten Ausführung des im Programmspeicher (8) enthaltenen, mit diesen Elementen (15) korrespondierenden Programmcodes (9) erzeugten Signalen (4, 5) verknüpft werden,

15

◦ wobei die Elemente (15) des Listings (2) des Programms (3) in einem ersten Teilbereich (11) eines grafischen Anzeigemittels (14) und die Signale (4, 5) in einem zweiten Teilbereich (12) des Anzeigemittels (14) darstellbar sind.

20

2. System nach Anspruch 1,

d a d u r c h g e k e n n z e i c h n e t ,

dass eine Markierung (20) eines Elements (15) des Listings (2) des Programms (3) im ersten Teilbereich (11) des grafischen Anzeigemittels (14) und eine Markierung (21) der mit diesem Element (15) verknüpften Signale (4, 5) im zweiten Teilbereich (12) des Anzeigemittels (14) vorgesehen ist.

30

3. System nach Anspruch 1 oder 2,

d a d u r c h g e k e n n z e i c h n e t ,

dass ein dritter Teilbereich (13) des grafischen Anzeigemittels (14) zur Darstellung mindestens eines Teils der Signale (4, 5) vorgesehen ist.

35

4. System nach einem der vorhergehenden Ansprüche,

d a d u r c h g e k e n n z e i c h n e t ,

dass die Schaltung (6) mit dem Prozessor (7), dem Programm-
speicher (8) und den applikationsspezifischen Hardwarekompo-
nenten (9) in einer Hardware-Beschreibungssprache beschrieben
sind.

5

5. System nach einem der vorhergehenden Ansprüche,
d a d u r c h g e k e n n z e i c h n e t ,
dass Mittel zum Anpassen des Systems an verschiedene Prozes-
sortypen vorgesehen sind.

10

6. Verfahren zur Verknüpfung und Darstellung von Signalen
(4, 5) einer Vorrichtung zur Hardware-Simulation (1) und Ele-
menten (15) eines Listings (2) eines Programms (3),

• wobei die Vorrichtung zur Hardware-Simulation (1) das Ver-
halten einer Schaltung (6) mit einem Prozessor (7), einem
Programmspeicher (8), welcher den Programmcode (9) des
Programms (3) enthält, und applikationsspezifischen Hard-
warekomponenten (10) simuliert und Signale (4, 5) als Er-
gebnis der Simulation erzeugt,

20 20 • wobei die Elemente (15) des Listings (2) des Programms (3)
mit den bei der simulierten Ausführung des im Programm-
speicher (8) enthaltenen, mit diesen Elementen (15) kor-
respondierenden Programmcodes (9) erzeugten Signalen (4,
5) verknüpft werden,

25 25 • wobei die Elemente (15) des Listings (2) des Programms (3)
in einem ersten Teilbereich (11) eines grafischen Anzeige-
mittels (14) und die Signale (4, 5) in einem zweiten Teil-
bereich (12) des Anzeigemittels (14) dargestellt werden.

30 30 7. Verfahren nach Anspruch 6,
d a d u r c h g e k e n n z e i c h n e t ,
dass ein Element (15) des Listings (2) des Programms (3) im
ersten Teilbereich (11) des grafischen Anzeigemittels (14)
und die mit diesem Element (15) verknüpften Signale (4, 5) im
35 zweiten Teilbereich (12) des Anzeigemittels (14) markiert
werden.

8. Verfahren nach Anspruch 6 oder 7,
d a d u r c h g e k e n n z e i c h n e t ,
dass in einem dritten Teilbereich (13) des grafischen Anzei-
gemittels (14) mindestens ein Teil der Signale (4, 5) darge-
5 stellt wird.

9. Verfahren nach einem der Ansprüche 6 bis 8,
d a d u r c h g e k e n n z e i c h n e t ,
dass die Schaltung (6) mit dem Prozessor (7), dem Programm-
10 speicher (8) und den applikationsspezifischen Hardwarekompo-
nenten (9) in einer Hardware-Beschreibungssprache beschrieben
sind.

10. Verfahren nach einem der Ansprüche 6 bis 9,
15 d a d u r c h g e k e n n z e i c h n e t ,
dass das Verfahren an verschiedene Prozessortypen anpassbar
ist.

11. Fehlersuchwerkzeug zur Verknüpfung und Darstellung von
20 Signalen (4, 5) einer Vorrichtung zur Hardware-Simulation (1)
und Elementen (15) eines Listings (2) eines Programms (3),
• wobei die Vorrichtung zur Hardware-Simulation (1) das Ver-
halten einer Schaltung (6) mit einem Prozessor (7), einem
Programmspeicher (8), welcher den Programmcode (9) des
Programms (3) enthält, und applikationsspezifischen Hard-
warekomponenten (10) simuliert und Signale (4, 5) als Er-
gebnis der Simulation erzeugt,
• wobei das Fehlersuchwerkzeug Mittel zur Verknüpfung der E-
lemente (15) des Listings (2) des Programms (3) mit den
30 bei der simulierten Ausführung des im Programmspeicher (8)
enthaltenen, mit diesen Elementen (15) korrespondierenden
Programmcodes (9) erzeugten Signalen (4, 5) aufweist,
wobei die Elemente (15) des Listings (2) des Programms (3) in
einem ersten Teilbereich (11) eines grafischen Anzeigemittels
35 (14) und die Signale (4, 5) in einem zweiten Teilbereich (12)
des Anzeigemittels (14) darstellbar sind.

Zusammenfassung

Verknüpfung und Darstellung von Signalen einer Vorrichtung zur Hardware-Simulation und Elementen eines Listings eines
5 Programms

Die Erfindung betrifft ein System und ein Verfahren zur Verknüpfung und Darstellung von Signalen (4, 5) einer Vorrichtung zur Hardware-Simulation (1) und Elementen (15) eines Listings (2) eines Programms (3) sowie ein Fehlersuchwerkzeug. Um eine gemeinsame Darstellung der Signale der Vorrichtung zur Hardware-Simulation und der Elemente des Listings des Programms zu ermöglichen, wird vorgeschlagen, dass die Vorrichtung zur Hardware-Simulation (1) das Verhalten einer Schaltung (6) mit einem Prozessor (7), einem Programmspeicher (8), welcher den Programmcode (9) des Programms (3) enthält, und applikationspezifischen Hardwarekomponenten (10) simuliert und Signale (4, 5) als Ergebnis der Simulation erzeugt, dass die Elemente (15) des Listings (2) des Programms (3) mit den bei der simulierten Ausführung des im Programmspeicher (8) enthaltenen, mit diesen Elementen (15) korrespondierenden Programmcodes (9) erzeugten Signalen (4, 5) verknüpft werden und dass die Elemente (15) des Listings (2) des Programms (3) in einem ersten Teilbereich (11) eines grafischen Anzeigemittels (14) und die Signale (4, 5) in einem zweiten Teilbereich (12) des Anzeigemittels (14) darstellbar sind.

FIG 1

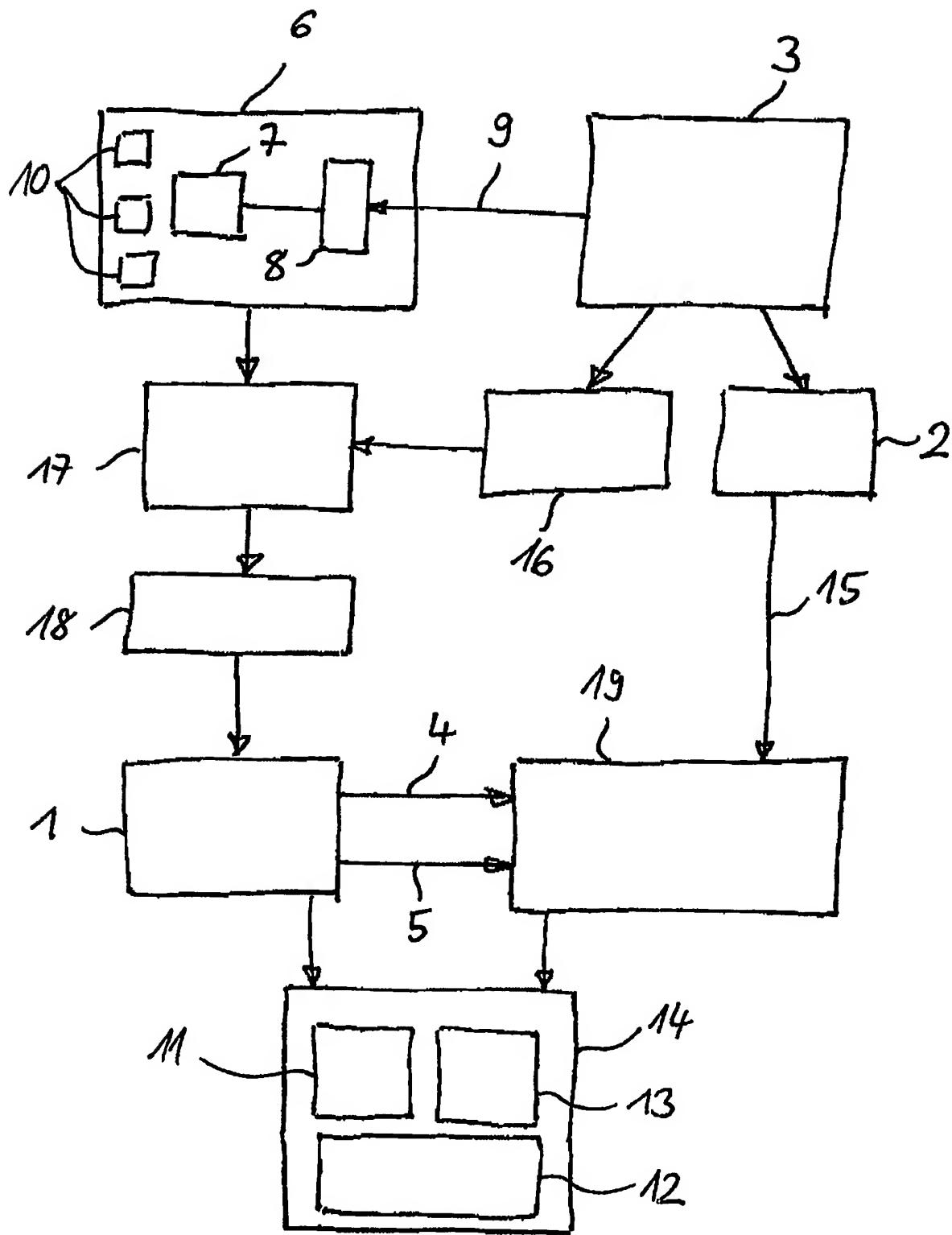


FIG 1

200308945

2/3

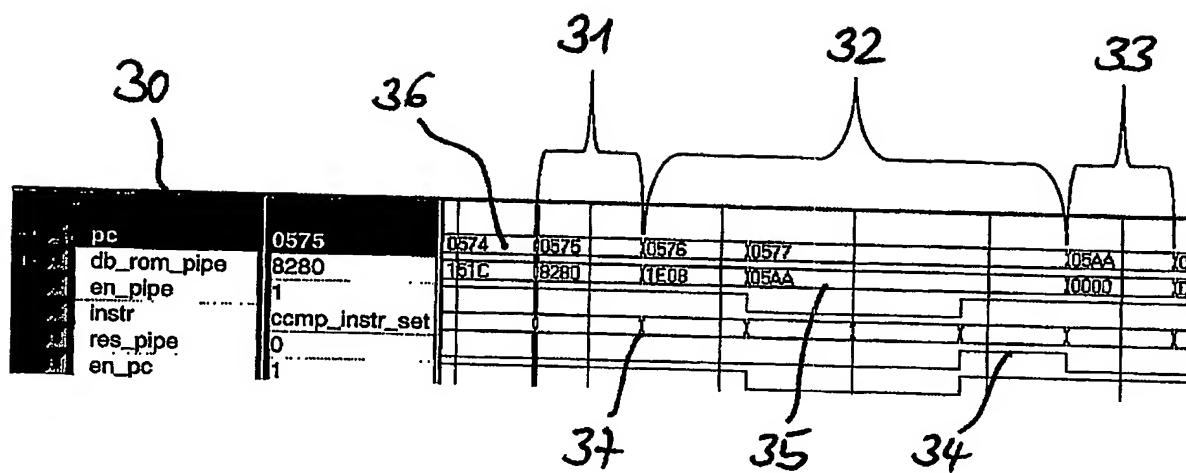


FIG 2

BEST AVAILABLE COPY

23

20

10

21

22

FIG 3